Intelligent Agents In Super Mario Bros

Kimiaei, Sep
22989891

Michlin, Nicholas
22719855

November 2, 2023

Abstract

This paper introduces the integration of reinforcement learning and rule-based systems into the popular video game Super Mario Bros (1983). Reinforcement learning and rule-based agents were bench marked against objective-based performance metrics. Early experiments with rule based agents using object detection imaging code provided direction in developing more complex agents that utilised deep learning and served as a baseline for comparison. With sufficient input pre-prossessing, Proximal Policy Optimization was found to provide the best results, taking approximately *1 500 000* training steps to consistently complete levels with high scores.

1 Introduction

Research and development into artificially intelligent systems has rapidly excelled in the public and private domain with popular research suggesting singularity may become the ultimate precipice [7].

Traditional decision-problem based environments such as Chess, Checkers and Go have been popular testing playgrounds for intelligent agents due to their optimal and scalable performance evaluation alongside iterative scenario modelling. However, the transition of artificially intelligent engines into complex video games has enabled further discussion into topics such as long-life learning, transfer learning, and action-exploration outside traditional human-based decisions [12].

Directed by Nintendo's Shigaru Miyamoto in 1983, the Super Mario Bros video game designed for the NES was a pioneer and turning point in the video game industry with its legacy amassing to approximately $25 billion dollars in revenue from the Super Mario franchise alone for Nintendo [1]. The addition of the Super Mario Bros engine into the publicly available domain of Open AI's gym environments has allowed for the research and development of this papers outlined agents in a complex environment. The Super Mario Bro's gym environment in particular offers the availability of assessing physics-based mechanics, fast-paced decision requirements, and rigid situational awareness all whilst only providing a limited view of the entire level.

In the past, reinforcement learning algorithms such as Deep Q-learning (DQN) [6], Advantage Actor Critic (A2C) [5], Trust Region Policy Optimization (TRPO) [8] and Proximal Policy Optimization (PPO) [9] have been used in the reinforcement learning game playing domain. Most commonly, such algorithms are bench marked against each other utilising Atari game playing environments provided by Open AI gym [2].

In the following sections, this paper discusses how a rule-based agent was developed as a baseline while reinforcement learning agents are bench marked against a rigorous objective-based performance metrics alongside loss loss evaluation and hardware usage analysis to outline their strengths and weaknesses.

2 Method

2.1 Requirements

The integration of the Super-Mario-Bros gym environment developed by Open-AI was done through importing the official stable-releases. All experiments were conducted on gym *0.26.2*, stable baselines *2.1.0*, torch *2.0.1* and cuda *12.1*.

For the rule-based agent, the integration of Lauren Gee's imaging solution into a state-space constructor (explained in Section 2.2) required image processing framework Open CV. It is important to note that our own imaging solution was also constructed prior to obtaining Gee's solution that used stand-alone HSV detection requiring no additional libraries.

The reinforcement learning methodologies outlined in

this paper were developed in a plethora of differing ways under iterative development. Our team did a hand-implementation of PPO that utilised pytorch (for the neural networks) alongside our state-space solution. Additionally, Stable Baselines 3 was used for the efficient and fast construction of a multiple agents for performance evaluation.

2.2 Inputs, Preceptors and State-Space

Made accessible through the gym's endpoints, game-specific sensory data such as Mario's coordinates, the default reward function, coins amassed, lives left, and the entire observation was made readily available removing the need for complex evaluative image-processing in the construction of a reward mechanism.

Due to the limited hardware and time available as students, a requirement for creating a reduced state-space using the available sensory was established. The need for this requirement was further embellished by the original large observation size of 255x255x3 (length, width and RGB color channels).

Discussion in the team lead to a summary of important sensory data that should be covered by Mario for a re-enforcement learning agent. This resulted in a [1x16] array contrived of these following key parameters:

- Mario's x and y positions (relative to the image).
- x and y distances of the 2 closest enemies ahead of Mario alongside the enemy nearest to Mario behind him.
- x and y distances of Mario to the closest pipe.
- x distance of Mario to the closest hole.
- *Width* of the closest hole.
- x and y distance of Mario to the closest question block.
- x and y velocity of Mario.

At first glance, this reduced state-space appears as an optimal advancement for training an agent. However as observed in Section 2.1, the missed critical information of this reduced state-space had drastic effects on the exhibited learned behaviours. This included the inability to jump from pipe-to-pipe as multiple pipes were not included, no situational awareness of blocks that Mario may collide with in the air and the reduced potential for adding in new entities.

In order to design a more comprehensive and flexible state-space, an approach was developed that would utilise the information of the whole observation whilst reducing the size. Inspired by the methodology used to preprocess Atari environments [6], a state-space compressing system consisting of *skipping frames*, *resizing the image frame*, *converting the image to gray scale*, *skipping and taking the max between the last 2 frames*, before finally *stacking frames* on one another.

A summary of the two different state-spaces is demonstrated in Table I.

Table 1: Table I: State Space Methodologies

Solution	Coverage	Type	Dimensions
Default	Entire observation	Image Array	255 x 255 x 3
Key Features	Predefined pre-pocessing rules	Numerical Array	1 x 18
Transformed	Transformed & reduced observation	Image Array	4 x 84 x 84

2.3 Rule-based

Dating back to the early days of artificial intelligence research, rule-based systems have played a pivotal role in various domains, including the first natural language processing chatbot ELIZA [11]. The team decided to develop a rule based agent before experimenting with deep learning techniques because it a allowed us to gain a comprehensive understanding of the rules and patterns that exist within the problem domain. As the rules were fully hard coded, it provided insight into the problems a deep learning agent would have to overcome and types of rules that it would have to learn.

2.3.1 Implementation

The original implementation of rule-based Mario involved the following steps.

Pre-processing observations This involved object detection which utilising imaging code provided by Gee. The imaging code was extended to utilise a *Bounding Box (BBox)* class which provided a simpler interface to perform calculations on. Fields include:

- X_{min} and X_{max} positions
- Y_{min} and Y_{max} positions (smaller value means higher)
- C_x and C_y representing central coordinates
- *width* and *height*
- $BBOX_{type}$ representing the object captured

Apply rules based on closest obstacle The team decided that only the closest obstacle or enemy should used to decide what rule to execute in a given situation.

Such was done by collecting all obstacles in front of Mario and sorting by X_{min}

Adding actions to action queue During development, we found that certain actions need to be repeated several times in a specific order. For example, the jump button needs to be pressed several consecutive times to get over a pipe or Mario requiring to sprint a few steps before making a jump over a hole. This lead to the idea of implementing a queue system to overcome such situations. More importantly, an action queue allowed Mario to get out of scenarios where he could get stuck, such as under pipes. The existence of an action queue also meant that imaging code need not be run if an action had already been lined up from previous steps, providing the agent with a tiny speed boost. This design feature influenced our design choice of frame skipping in deep learning iterations later on.

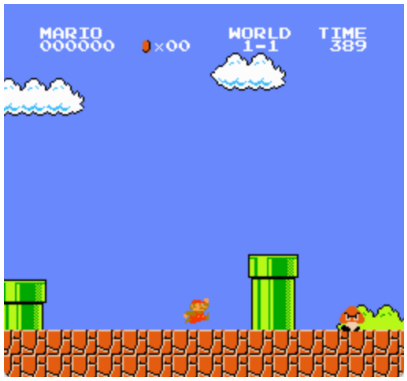
2.3.2 Scenarios and Rules

Rules were developed for the scenarios described in 1. By default, Mario holds down the sprint button while there are no obstacles in view. Upon encountering a single enemy as in 1a, Mario performs a jump action when the distance is less than a block size (16 pixels). For multiple enemies as in 1b, a jump action is followed by a series of NOOP actions to prevent Mario from running into the following enemies. As for pipes, described by 1c, Mario makes a run up and holds down the jump button for a duration proportional to height of pipe. A similar strategy was adopted for holes, as described in 1d but with additional run up steps added to the action queue. Lastly, Mario performs a jump action when within a block size of a stair block as described in 1e. For pipes and holes, it is possible that Mario gets stuck when too close. In the event of getting stuck, Mario is freed by adding several left actions to the action queue.

(a) Single enemy

Figure 1: Scenarios rules were developed for

(b) Multiple enemies



(c) Pipes

(d) Holes

(e) Stairs

Figure 1: Scenarios rules were developed for

2.4 Proximal Policy Optimisation

Reinforcement learning techniques were explored in the following order: Q-learning, Deep Q-learning, Double Deep Q-learning, Proximal Policy Optimisation; all experiments started with the simpler state space (result of using Gee's imaging code).
We found that Q-learning resulted in a very large Q-table, which led to the team 'bucketing' distances to further reduce the state space. As Q-learning is a form of value iteration, the agent was slow to converge to a value function which led to optimal actions being taken. Deep Q-learning provided slightly better results however it was found that Mario rarely progressed beyond the second pipe. We believe the reason for this is largely due to DQN's being an offline reinforcement learning algorithm and the existence of a fixed size memory replay buffer. As Mario spends a substantial amount of time learning to progress past the second pipe, the replay will contain a large number of states where Mario is stuck behind the pipe, leading to a large imbalance of scenarios the agent can optimise on. Further, should Mario learn to get past the second pipe, there is a low chance that newer states experienced will be optimised due to the imbalance in the memory reply buffer. Choosing an appropriate exploration rate ϵ and decay also posed a challenge as it was found that a larger value prevented Mario from getting past the first pipe and a low value would discourage exploration after choke points. We believe DQNs are sufficient in environments such as CartPole however the offline learning nature and the existence of a fixed memory replay buffer severely inhibits learning in environments where a certain level of progression is required in order to be exposed to new states such as in Super Mario Bros.

Online policy learning techniques such as A2C [5], TRPO [8] and ACER [10] exist however the team decided to proceed with PPO. This decision was based on the notion that PPO being the most recent and builds on the advantages of previous works above. Inspired by A2C, PPO adopts an Actor Critic architecture where the Actor network is responsible for selecting actions based on policy π_s and the critic is responsible for approximating Value function V_s. A2C alone however is very sample inefficient as it is a form online learning algorithm where experiences are used once with no memory replay buffer. To overcome this, PPO employs a training loop that involves a roll out phase where the current policy is used on the environment to collect data, followed by a training phase where the network updates on the collected data in mini batches based on a predefined number of epochs; providing the best of both offline and online policy gradient methods. In addition, PPO takes the idea of trust region methods from TRPO but modifies it to use a simpler clipped objective which takes the lower bound of a generalised advantage estimation score. This results in a model that conservatively improves its policy while retaining the ability to severely punish poor actions. In terms of exploration vs exploitation, PPO policies are represented as a probability distribution over possible actions. We included the optimisation where hidden layers utilise an orthogonal initialisation to increase initial exploration.

2.5 Reward

2.5.1 Reward Function Formulation

The default reward provided by *gym-super-mario-bros* includes rewarding Mario for moving in the positive x direction, penalising per difference in in-game seconds per frame and imposing a large death penalty. This was further extended by rewarding Mario based on in-game score which can be obtained by collecting coins, power-ups, killing enemies and completing a level. The additional reward collected through in-game score is then scaled down by a factor of 40 so that the score is not significantly superceding the other reward metrics.

2.5.2 Additional Environment Wrappers

With inspiration from how the Atari environment for breakout was preprocessed [6], we included an episodic life end wrapper to which provided a clearer view on cumulative episodic return; aiding in value estimation.

3 Experiments

3.1 Establishing Performance Criteria

Comprehensive and exhaustive assessment criteria was developed to assess the quantitative performance of each agent in conjunction with the prominently observed attributes.

The following experiments were performed:

1. Exhibited policy and value loss alongside mean cumulative reward and explained variance.
2. Time and reward obtained for rule-based vs reinforcement-learning agents.
3. State-space Performance Evaluation.
4. Transferability.

In addition to the above, the following visualisations to assess behavioural strengths and weaknesses were performed:

1. PPO Policy & Decision Making Process Visualisation.
2. Innovation Snapshots.
3. Hardware Reporting.

The results of these findings are presented in the proceeding section.

3.1.1 Reinforcement Learning Model Selection

Prior to analysing performance comparisons across rule-based and reinforcement-learning agents, it is important to develop an optimal reinforcement-learning model that can act as a suitable benchmark. This entails finding the suitable amount of required training data alongside a sufficient input state-space.

Due to the behavioural nature of reinforcement learning agents, it is unreasonable to assess the performance of a model based purely on analysis of it's loss metrics. As a result, the first experiment involved the generation of policy and gradient loss alongside the cumulative reward and explained variance metrics in order to find an optimal amount of training data.

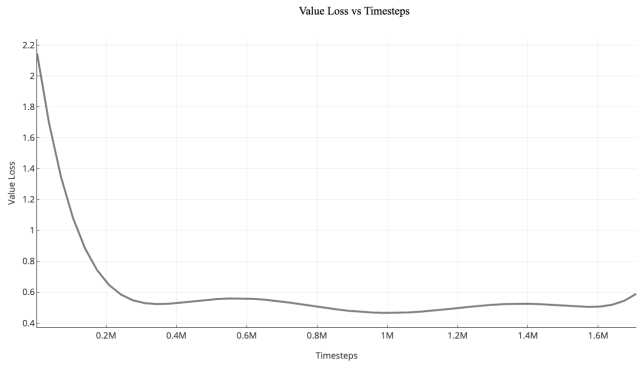


Figure 2: Smoothed PPO Value Loss During Training

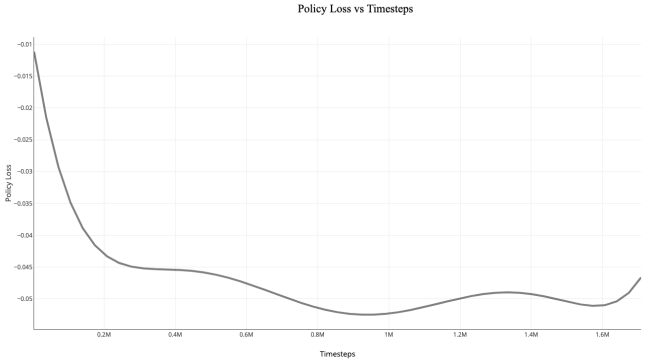


Figure 3: Smoothed PPO Policy Loss During Training

The value loss presented in Figure 2 corresponds to the model's expected value for Mario's state versus the observed actual reward returned from the gym environment. As expected in reinforcement learning agents, the value loss decays exponentially until a point of convergence. In our Mario simulation, this point of convergence was at a value of ~ 0.53 after ~ 1.6 million time steps. Despite this, sporadic jumps in loss were seen occurring throughout the training cycle (in the raw non-smoothed data) with some recorded as ~ 3 times higher than the average at the time. This can ultimately be explained by the new states the agent discovers as there is no informative policy to base its predictions correctly on. Policy loss is dependent on the rate of change for the formed policy throughout training, and was modelled in Figure 3. A similar relationship was observed in Figure 3, indicating the policy loss exhibited the same generalised decaying form as well as exhibiting the same high jumps due to new novel states.

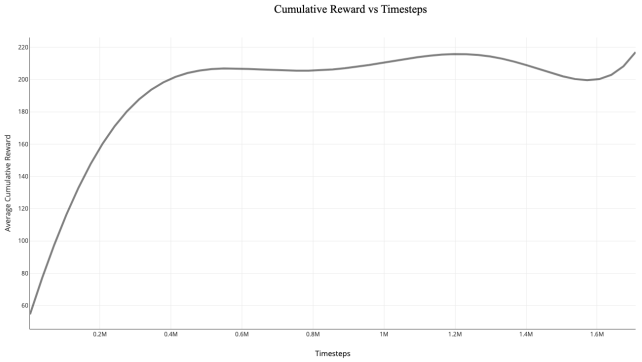


Figure 4: PPO Cumulative Reward During Training

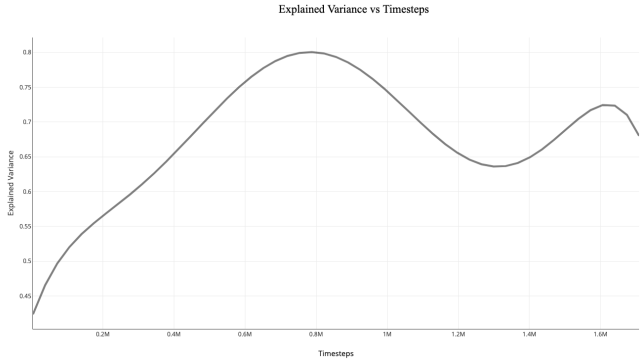


Figure 5: PPO Explained Variance During Training

As mentioned previously, analysis of only loss metrics is not enough when deciding upon a training region. The average episodic reward that was received upon acting on the policy throughout training is observed in Figure 4, with a linearly increasing region for the first 500K time-steps followed by an asymptotic region for the remainder of the trained dataset. The episodic return saw an average return of 220 from our team's defined reward function within this episodic region. To verify that this reward was sufficient, the game was launched with training data ranging from 1.2-1.7 million with which it was able to successfully finish the level.

In a similar fashion, the explained variance increases in a linear manner initially but then a major dip can be noticed. The explained variance is a measure of the policy's strength when accounting to variation in the data. As the policy began to learn, it was often only able to get to the second pipe and thus its respective region of training was strengthening only up until that point. After having learnt that area, the agent was able to see holes and new parts of the level for the first time which correlates to the sudden dip at 900K time-steps before a new strengthened region after approximately 1.6 million time-steps was developed.

Taking points of salience from the cumulative reward, explained variance alongside regions of loss convergence allowed us to be confident that 1.6 million time-steps would be an appropriate amount of time-steps. 1.6 million time-steps ensured that the policy was strong in regards to predicting the obtainable reward from performing certain actions without requiring further hours of testing.

3.1.2 State-space Performance Evaluation

Outlined previously, an important initial design parameter that our team needed to configure was the desired input state-space.

Summarised in Figure 6, an obvious trade-off for Gee's imaging code versus the transformed observation would be the size of information versus the quality and breadth of the information fed into the Neural Network. To test this hypothesis, an experimental procedure was set up that benchmarked the PPO agent on both differing state-spaces for 100K time-steps (1-1 shortened to the first hole with a custom wrapper) that assessed their respective performances.

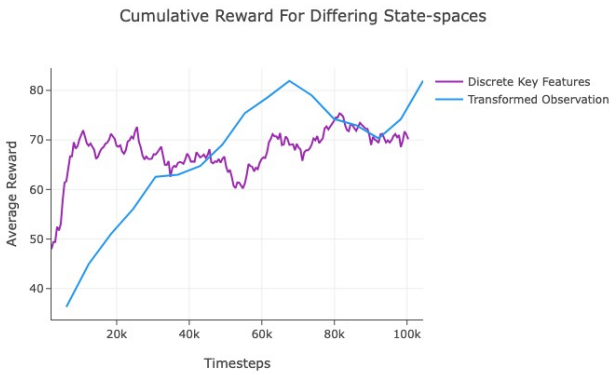


Figure 6: State-space Performance Test

The transformed observation encompassed the entire game image (alongside gray scale and resizing transformations) whereas the the Discrete Key Features state-space used Lauren Gee's object detection framework to produce a single dimensional array consisting of key objects discussed in Section 2.2. During the initial training period, the discrete key features method performed better most likely as a result of only having to learn what the appropriate action is when a Goomba is nearby. Within this initial region, the transformed observation performed comparatively poorly which we concluded was a result of the large time required for the model to correctly identify the importance of certain elements within the large image.

After this initial period, the transformed image observation began to excel. Within our team we marked this region as the point of "confidence" as it seemed to have finally learnt the damaging effects of encountering a Goomba and becoming stuck at the base of a pipe. The additional information contained in the transformed image enabled Mario to also explore the possibly of performing actions such as jumping from pipe-to-pipe if the distance was within a possible range. These newfound rule-sets explain the increase in performance for the transformed image state-space compared to the seemingly constant results obtained from the discretized key feature set.

For these reasons alongside Section 4.3, our team decided to use the transformed observation of the entire frame for training our reinforcement learning agent. Furthermore, this experiment outlined an advantage reinforcement-learning techniques inhibit which is the ability to feed in the entire observation when the developers are unsure of what the key caustic elements are. In a rule-based agent, the input space would have to been carefully selected in accordance to the rule-sets requirements inducing a more complex and less adaptable code framework.

3.1.3 Rule-based vs PPO Quantitative Performance Comparison

After selecting the training data derived from 1.7 million steps, rudimentary tests between the rule-based and reinforcement learning agents can be performed that assesses their relative speed and cumulative score obtained for complete runs of Level 1.

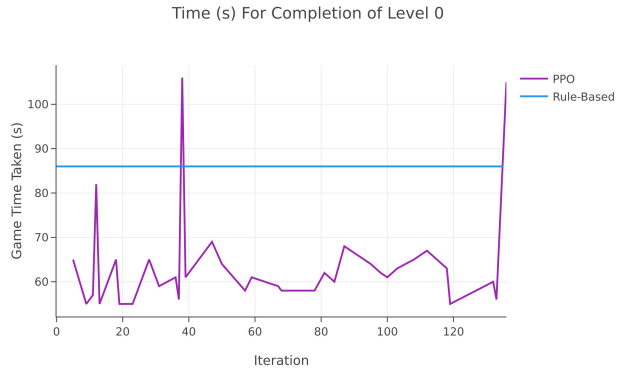


Figure 7: Multi-agent Speed Test

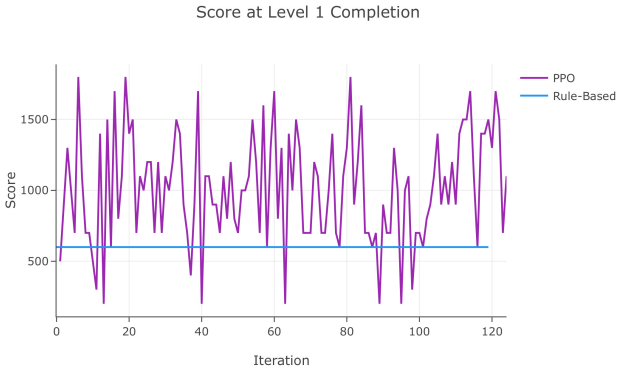


Figure 8: Multi-agent Score Test

Observed in Figure 8, the PPO agent trained on 1.7 million time-steps consistently returned an average reward of $\sim$ 990 (with a recurrent peak of 1800 if ran determinsitically) as opposed to the consistent 600 returned by the rule-based methodology. This presents an approximate 200% increase in score obtained alongside a 23% decrease in time taken (87s down to 67s) to achieve that score whilst completing the first level.

These results provide the first insight into the potential weaknesses and strengths of both methodologies. The poor relative performance of the rule-based methodology demonstrates the common problem of immutability in rule-based agents as the design is predefined. As evident, the rule-based methodology would provide results deemed optimal by our team (comprised of two computer science students in a short time frame) with no potential to improve its performance. On the other hand, a PPO agent was able to derive higher performing and more complicated rule sets owed to it's complex mechanisms outlined in Section 2.4.

However, adaptability and the ability to find more optimal solutions does not come without a price. Although both agents were able to complete the desired level, the success rate of the rule-based was extremely high (95% from multiple runs) compared to differing outcomes produced from training PPO agents. This stability in results alongside complete transparency in the decision making process is one of the main drivers behind the adoption of rule-based algorithms in industry [4].

3.1.4 Adaptability to the unknown

In the realm of decision-based problems, an important design criteria is an agents ability to perform on a new unseen environment. In order to test this, a test pipeline was configured consisting of measuring the performance of both agents on Level 1-W1 and subsequently Level 1-W2. The results of this experiment are observed below:

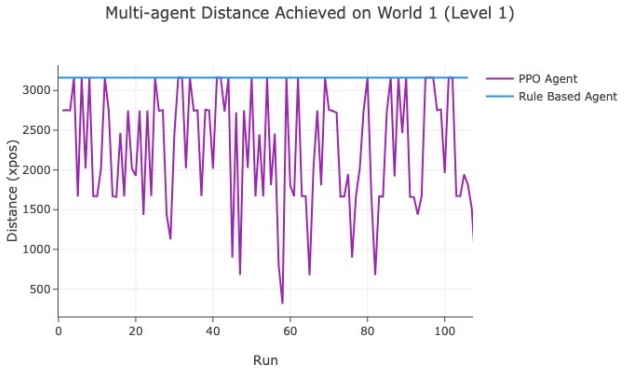


Figure 9: Multi-agent Distance Test (World 1)

Figure 10: Multi-agent Distance Test (World 2)

Designed for Level 1 - World 1 there is no surprise that the Rule-based agent performs strongly with stability in World 1 - Level 1. However, transferring the agent to an unseen environment such as World 2 - Level 1 evidently induced struggle. This is accentuated in Figures 9 and 10 that shows a comparative struggle for the rule

based agent to perform on the next level. To explain this lack of transferable performance, the first reason we thought of was the existence of new scenarios that were not accounted for in the rule-based methodology. The PPO was able to mitigate the loss of the 'unknown and novel' scenarios by using its respective neural net to approximate the appropriate actions for the new scenarios.

This reasoning also further signified the importance of why we decided to not use our original implementation of a Q-table consisting in the form of a look-up table with discrete values. In a game like Mario, there are too many potential discrete states that any algorithm that did not use a function approximator (such as an NN) would fail on any transferability tests unless it was trained on a significantly larger data set (entirely encompassing)

After investigating further into why the PPO was much stronger in transferability tests, we realised a significant factor was to do with the schematics of the neural net. As our team had used a CNN (convolutional neural network) to initiate the layers of the network, general features such as edges and prominent areas were detected early on. This generalisation of features made possible in the neural network ensured that new images passing through the convolutional layers would be approximated by the function based on their key components rather than the whole image.

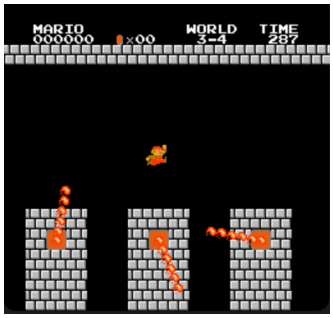


Figure 11: Position x:386 on World 3 (Level 4)

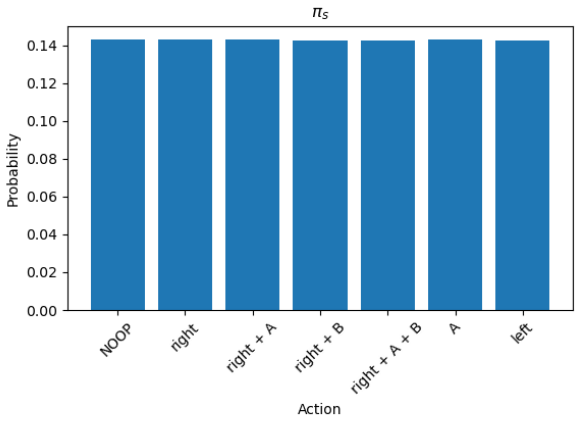


Figure 12: Initial Policy Probability Distribution

4 Visualisation and Debugging

When conducting experiments to garner insights into the decision-making process, a series of visualisations were developed in order to provide assistance in the real-time monitoring and evaluation of our scientific methods.

4.1 Policy Modelling

The first issue we encountered was the struggle to gain an understanding into the policy as the PPO model began to train. For a rule-based approach, this was very clear due to us creating the decision making process ourselves which is one of the core strengths provided for rule-based methodoology. In order to provide the same transparency for a deep-learning environment, we created a live histogram that displayed the probabilities (referred to as logits) of each action being pressed at a specified state. To demonstrate this, a position of (x: 386) was chosen on Level 4-W3 with the results attached below.

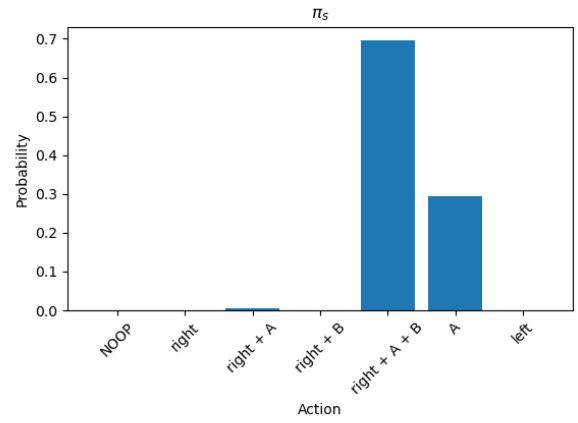


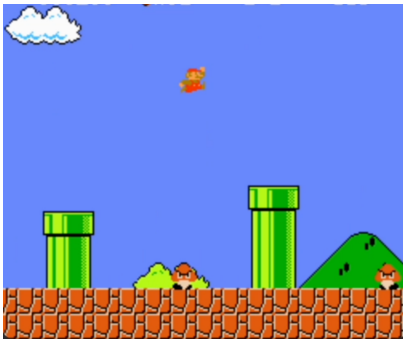
Figure 13: Policy Probability Distribution after 1.7 Million Time-steps

As expected, the policy function in Figure 12 demonstrates a uniform distribution in-line with the initial orthogonal setting our team had chosen for the model. Observing the change in policy after 1.7 million iterations in Figure 13 demonstrates the decision making

process the PPO has been able to derive in great detail. Figure 13 demonstrates high non-zero probabilities for performing right + A + B (jump & sprint), A (jump) and right + A (jump and move right). Performing a side-by-side comparison of these expected actions against the scenario presented in Figure 13 deems that the PPO is indeed performing optimally at this stage due to those actions being valid moves that can be performed at that state (with the safest and most rewarding being *right + a + b*. Although convergence was visible in our training dataset, it is not entirely implausible for a scenario of single-action domination occurring given a very large amount of time-steps for training.

4.2 Innovation

As previously discussed, one pitfall in the Rule-based methodology was the inability to innovate and explore novel ideas beyond our team's expectations. Whereas the PPO agent was able to present very unique and interesting solutions to some of the complex environments it encountered.
In order to document these, our team constructed a pause system that enabled us to force the policy to perform a NO-OP (no operation) on demand in order to document the interesting behaviours it preformed. The following images were obtained:



(a) Pipe-to-Pipe

(b) Multi-Action *(jump + down)*

Figure 14: Notable Observed Innovations

(c) Run-up

Figure 14: Notable Observed Innovations

Summarising these images, the policy was not only able to identify an optimal action of jumping from a pipe to another pipe (thus reducing walking distance and time) but also successfully identified the ability of ending a jump shortly (pressing down immediately) and performing a run-up to clear a pipe in case it got stuck underneath it. Although this visualisation provided an amusing gaze into the PPO agent's mind, it more importantly emphasised the double-edged sword of improved yet unexpected behaviour obtained from reinforcement learning models.

4.3 Hardware & Storage Debugging

Performing these simulations as two university students with limited hardware was a major motivation behind the previously outlined training benchmarking and state-space optimisations we performed. In order to assist with those experiments and further explore the domain of capability within our design, we added FPS and storage monitoring as the training was performed.

Table 2: Hardware/Storage Profiler Results

State-space	FPS	Time taken (100K time-steps)	Model Size
Transformed Observation	~200	13.89 minutes	20.50 MB
Discrete Key Features	~ 6	4.63 hours	158 KB

This further justified our decision to use the transformed observation as opposed to discretisizing the image into a single array consisting of key features. This debugging output signified the large bottleneck that was induced by incorporating Lauren Gee's imaging code due to the large amount of image processing required with a total time for completion being ~ 21 times longer. Furthermore, comparing this output to the performance difference in Section 3.1.2 signified the trade-off for storage and performance when deciding upon a state-space. It turned out that simply making a state-space smaller might make the model file smaller (and potentially the processing time) but ultimately shortens the skill-ceiling and must be decided upon on a case-by-case basis. In

regards to both agents, they ultimately both suffer from the choice of state-space as a rule-based agent's design methodology would be limited to its permitted inputs as that is often the only input within its state-machine.

5 Suggestions for Future Works

It is worth noting that models above have been trained on a per level basis. Methods to train Model that is able to complete levels in sequential order would be an interesting talking point.

5.1 Hyper Parameter Fine Tuning

Our team found that Reinforcement learning algorithms are very sensitive to hyper parameters. Reinforcement learning in the Super Mario Bros environment involves solving a large sequence decision problem; further experimentation with hyper parameters would provide a better understanding on solving such problems and how algorithms react in long sequence problems.

5.2 Reward Parameter Fine Tuning

We suggest experimenting with different reward functions to alter Mario's playing style. Possible play styles our team has thought of include:

- Zen Mario where no enemies are harmed in completing the level
- Coin Mario where coins are heavily prioritised
- The Floor is Lava where Mario spends more time on platforms such as pipes and floating blocks

5.3 State Space Pre-Processing

As discussed above, state space pre-processing was a major bottleneck in our development process. Currently, our models with image as inputs utilise convolutional neural networks (CNN) to understand mario's environment. We suggest experimenting with the image processing by substituting the existing CNN with a pretrained image processing model to see if performance can be improved.

For a simpler state space, an Object detector such as Meta's detr-resnet-50 [3] could be experimented with in place of Gee's object detection code code.

6 Conclusion

Through the initial development of a rule-based agent, we were able to construct a suitable baseline for machine performance on the Super Mario Bro video game. Building upon this, we constructed a multitude of vanilla reinforcement-learning agents (Q-learning and DQN) before constructing the final Proximal Policy Optimization (PPO) agent. In order to assess and optimise both the rule-based and PPO agent, a series of comprehensive experiments and supporting visualisations were performed and their results not only highlighted optimal parameters but further divulged the key differences between rule-based and reinforcement learning algorithms alongside their respective strengths and weaknesses.

References

[1] Aaron Astle. Nintendo's mobile future unclear despite "tripled" revenue.

[2] Greg Brockman. Openai gym. 5 June 2016.

[3] Nicolas Carion. End-to-end object detection with transformers. 28 May 2020.

[4] Sujata Dash. Metaheuristic-based hybrid feature selection models. 2018.

[5] Volodymyr Mnih. Asynchronous methods for deep reinforcement learning. 16 June 2016.

[6] Volodymyr Mnih. Playing atari with deep reinforcement learning. 19 December 2013.

[7] Darren Orf. Humanity may reach singularity within just 7 years, trend shows.

[8] John Schulman. Trust region policy optimization. 20 April 2017.

[9] John Schulman. Proximal policy optimization algorithms. 28 August 2017.

[10] Ziyu Wang. Sample efficient actor-critic with experience replay. 10 July 2017.

[11] Joseph Weizenbaum. Eliza—a computer program for the study of natural language communication between man and machine. Jan 1966.

[12] Peter Wurman. Improving artificial intelligence with games.